General Comments on Scheduling, Resource Allocation, and Storage Management

F.J. Corbato
J.H. Saltzer
May 6, 1965

The following remarks are an attempt to summarize discussions at Project MAC and Bell Labs. Some of the active participants in these discussions have been: V. Vyssotsky, H. MacDonald, J. Ossanna, D. Eastwood, at Bell and E. Glaser, R. Graham, and R. Daley at MAC.

In general, resource allocation and scheduling in the time-sharing system for the GE 636 should be more elegant and simple than on the present MAC 7094 system. In particular, the strategy of usage of the new machine must be such that all processes are multi-programmed in a general way by a supervisor program using queues. By using queues it is possible to separate the policy setting mechanisms from the mechanics of servicing queues. Distinct policies may establish the relative priorities of distinct entries in a queue but the servicing mechanism does not have to be aware of these individual policies. When viewed in this light the scheduling algorithm becomes really a priority mechanism for the tasks that the system must accomplish. Thus when a processor becomes idle it should go to the list of tasks ready for a processor (i.e. tasks with at least an initial page of program in core) and assign itself to the highest priority process. Similarly, when an I/O controller becomes available, the priority of I/O requests in the queue for that controller has already been determined by the scheduling algorithm. Thus, the I/O controllers and the CPU's take on the simple symmetry which is required in a multi-programming system.

As is hinted already, it is proposed that the 636 system be implemented with a full page-turning philosophy in which a process is eligible for a processor as soon as the first program page is loaded into core memory. Since it is probable that another page will be needed quite abruptly, page turning implies that a large number of processes are available for the processors to dart among. Clearly, the overhead of switching a processor from one process to another must be small compared with typical process running times. In addition, no attempt should be made in this system to distinguish system processes from user processes but rather the page turning mechanism, on the basis of activity alone, should determine which program pages should remain in core memory. Of course, exceptions will probably be necessary for certain processes, such as those which answer timing-critical interrupts. Similarly the page turning algorithm itself will probably require permanent presence in core memory.

The Page-Turning Algorithm

To explore this page-mechanism further, a reasonable implementation would be to keep a table in the supervisor which consists of a cell per page where each cell acts as a counter of inactivity of the page. Whenever a fixed interval of time, which should be a parameter of the system occurs, e.g. 200 m.s., the supervisor should inspect all page activity bits, increment the counters of those pages which have not been active in the interval, and reset to zero the counters of pages which have been active. Whenever the supervisor needs a fresh page in core memory it should refer to a free page list. Whenever the free page list becomes low the supervisor should trigger a page-eviction algorithm which removes from core memory those pages which have been least active. The eviction algorithm should be a function of inactivity sampling rate, inactivity threshold, free-page refreshment levels, etc., all of which may be dynamically adjusted with some damping algorithm. The

supervisor then automatically defers to secondary storage inactive pages, taking into account that pages which have not been modified since loading need not be written out again. It is expected that the supervisor will initiate secondary storage I/O tasks in the same manner as a user program so that as much as possible the supervisor resembles a user's program

A special problem is posed by the fact that the 636 hardware paging mechanism may be operated with 2 page sizes, 64 word blocks and 1024 word blocks. One possible strategy for this problem is to have two completely independent page-eviction algorithms operate in the two types of memory. If one of the two page evictors becomes "overworked" before the other it may become worth while to juggle the amount of memory run in each page size mode. Breaking up a 1024 word page is no problem, but a special garbage collector must be used to collect enough 64-word pages to make a larger page if memory juggling is done dynamically. This garbage collector may work in one of two ways after locating a 1024 word block with a minimum of active 64-word pages contained in it:

1. Move each active page with load and store to a different 64-word block taken from the free storage list.

2. Eject the active pages from memory as though they had become inactive, and depend on the page fault mechanism to retrieve them as soon as they are needed.

In both cases, the appropriate page descriptor table must be modified, and a check must be made to insure that some other processor is not actively taking instructions from the page being moved or ejected. The relative attractiveness of these two techniques depends on the speed with which they can be accomplished.

Other Supervisor Housekeeping Functions

The supervisor should basically be programmed such that its various processes operate on an independent asynchronous basis. A process may operate recurrently by going to "sleep" for a specified period of time and being reactivated by a clock interrupt. To maintain order among several such "sleeping" processes, there must be a master timer routine which contains a wake-up list of all processes that wish to be called upon. The basic information a process gives to the timer routine is that of the time and location at which it wishes to be restarted. When a process goes to sleep the timer always resets the active hardware clock interrupt to the time that it must next wake any process up. (This procedure resembles that used in the program MITMR at M.I.T.)

It is important to recognize that in the 636 time-sharing system the secondary storage mechanism will be handled in a hierarchial fashion without explicit control by the user. The user's process will interface the secondary storage mechanism using calls wherein files are referred to by symbolic name and by relative address, much as in the MAC 7094 system today, but without explicit knowledge of the physical location or the media upon which the material is stored. The secondary storage media should be arranged in a hierarchy according to performance and capacity with high speed drums on top, discs next, data cells next, and tapes perhaps last. The supervisor will include a "demon" process whose sole function will be to move files up and down the hierarchy according to activity. At least initially it is expected that up-and-down file movement will be sufficient although clearly there can be more elaborate reservation schemes. Again files belonging to system processes should remain on the high end of the hierarchy only on the basis of activity. Thus, after some period of time when the high-speed drum is becoming full, the most infrequently used files should be copied down to

the disc and so forth. Similarly, when a user refers to a file which has been inactive for a long period of time it should be copied up the hierarchy. The advantages of such an automatic system should be relatively obvious, and moreover it should be clear that great flexibility of equipment configuration is possible. For example, one may be able to remove a defective unit on a particular day from within the hierarchy, increase capacity, etc. The full details of how a procedure similar to this one should operate are covered elsewhere in the work of M. Bailey at Project MAC. Suffice it to say that a great deal of care and thought must go into considering the separate issues of 1) backup, 2) maintenance, 3) retrieval from accidents, and 4) retrieval from inactivity.

Finally, in addition to the secondary storage hierarchy, there must also be an interface to detachable storage, such as, tapes, disc packs, etc. The user interfaces with detachable devices with exactly the same kinds of calls as for normal secondary storage, but clearly the attachable devices are not in the hierarchy. Moreover, the user must have the facility for referring to non-standard detachable units which must be read using every trick of the physical reading device.

An important part of the file hierarchy system is that every file which is created and which has existed for some period of time is backed up with a duplicate copy on lower class storage. The backup mechanism would typically come in to play when the user logs out after a console session. From an organizational point of view it should be clear too that core memory may be treated as the highest member of the storage hierarchy. Thus, if possible the directory system and eviction algorithms used on the secondary storage media could also be available for core memory. (If they do not become too ponderous.) This viewpoint may become quite important as the number of processes kept in core memory becomes large, (e.g. 100).

Monitoring, Scheduling, and Time Accounting

Three very similar functions must be kept distinct: (1) monitoring of resource usage, (2) scheduling on the basis of resource usage, and (3) "time accounting" or charging the user for his resource usage. Since all three of these functions have a basis in the use of hardware, a fundamental piece of information kept for each user of the system must be a resource usage list. In this list, the supervisor might keep track of (for example):

Tc	CPU usage time
Ti	I/O controller usage time
Tt	Console usage time
Tm	Memory usage time weighted by pages used
Tp	Program usage time (for proprietary programs)
Ts	Secondary storage space-time usage

In general, this list reflects the amount of service the system has provided the user, in cold hard quantities. By examining this list for every user one can answer questions such as "Are the I/O controllers being used more than predicted?" The actual list used might break some of the system resources down into finer divisions, or add new classes of system resources.

It is proposed that the priority scheduling algorithm used be roughly analogous to the one used on the 7094 time-sharing system but with some important generalizations. A multi-level priority system with several levels and a floor-level still appears to be a reasonable technique. This mechanism should be serviced periodically with the period a dynamically adjustable system parameter. The current algorithm assigns central processor time on the basis of the formula t = (2.P.j)*q where "j" is the priority level of the process, and "q" is a time quantum. This time is assigned to the first member of the highest priority (that is lowest level number) queue. Jobs enter the queue with a small level number (adjusted for program size) and

gradually cascade to higher levels after using up the allotment at each level until they finally finish an interaction. Periodically the queue structure is inspected and pre-emption is allowed whenever an equal or higher priority process is available and a current user has run as many quanta as the pre-emptor will be granted; Otherwise pre-emption is deferred and the current process is continued. Because the present scheduling algorithm does not take into account I/O channel usage, core memory space etc., it is not in proper form. To generalize the present algorithm it is proposed that in place of CPU time as the basic variable which causes processes to cascade down the queue structure, that a resource usage function be substituted. This function is defined as a sum of resource time usages with cost coefficients. The level number at any time is defined as $j = \log_a r$.

The resource usage function is then defined as a weighted sum of the individual resources which the user has used during this interaction. For example,

$$r = A1 * \Delta Tc + A2 * \Delta Ti + A3 * \Delta Tn$$

might be a possible function, with the A's constants, and the ΔT's the amount of resource used since the beginning of the interaction. By properly adjusting the coefficients it should be possible to make the system fight back in the right places and give best use of its limited resources. (A presumption here of course, it that the heavy user of resources will not really notice that fraction of time used to service with higher priority the small user of resources.) One remark based on intuition and experience perhaps is in order concerning tuning of the scheduling algorithm. In general, it would appear wise to tune the algorithm such that most jobs operate on a first in, first out basis and only if they use an "unusual" amount of resources do they go to a lower level. (This is how Project MAC is currently tuned.)

The third function based on resource usage is accounting for user charges. Again, the user is called to account for his resource usage by a price function, P, which might be defined as

P = B1*Tc + B2*T1 + B3*Tm + B4*Tp + B5*Tt + B6*Ts

The B's are now adjustable coefficients, possibly identical to some of the A's in the priority resource usage function. It is more likely, however, that B1 and B2 will have different values during "prime-shift" time than late at night.

In the same general area of time-accounting, there is the subject of quotas. It is assumed that the user is charged for whatever resources he actually uses, according to the P function. Quotas are not necessarily rigid stops but rather trigger thresholds for special supervisor procedures. Depending on which quota has been exceeded, any of several possible policies may be followed. The user may be queried about his desire to proceed (and spend more than planned) or perhaps an automatic usage reduction mechanism may come into play such as the demon which moves files among secondary storage devices. Another possibility is that exceeding a certain quota requires the authorization of another user. A user (or his superior) should be allowed to adjust any of his quotas at any time, or declare a quota meaningless if necessary. A short list of possible quotas is the following:

1. Resource usage (in the priority sense) for a single interaction.
2. CPU time used for a single interaction.
3. Additional secondary storage added by a single interaction.
4. Number of pages of core memory.
5. Total space used by this user on drum storage.
6. Total space used by this user on disk storage.

7. Total dollars spent by this user this week.
8. Total time spent using a console this week. (Useful for classes.)

Classes of Service

There are, from a user's point of view, at least three distinct classes of service which he would like to obtain from a general purpose computation facility:

1. Interactive console service - the class of service now offered by the MAC 7094.
2. Console-less service - similar to "background" on the 7094.
3. Guaranteed access, for special real-time experiments.

We have already indicated the nature of a scheduling algorithm for interactive console service. The second class of service fits into the same general framework very easily. It is assumed that a console-less job is initiated at a console. However, there is no desire to tie up (and pay for) a console on long jobs and on the other hand it is already observed that no user cares to wait for more than 15 seconds for a response to an interaction. In fact, it is observed that the largest satisfaction arises when a user can be scheduled on a predictable basis (barring machine failure) for the return of a run. This goal is not even achievable today in the usual batch processing environment but still it seems reasonable to ask that it be achieved within the highly organized system we are proposing.

To accomplish the scheduling of console-less jobs in this environment the following technique is proposed. A hypothetical division of the machine resources is made into streams of service, for example, perhaps 85 percent of the machine is allocated to the foreground users at consoles, a remaining 10 percent is allocated to non-interactive jobs at

a certain cost figure and perhaps another 5 percent is allocated for non-interactive jobs, on a premium cost basis with, say, 50 percent higher price. It should be clear that all resources have dollar prices in this environment. There also may be a category of absolutely bottom priority jobs which will run on an unpredictable termination basis. Presumably such service would have a very cheap rate (e.g. as with electric power and the aluminum industry). In any case, whenever a user wishes to leave a job of known duration (or with a limit of what he wishes to pay in case the job does not terminate properly), the supervisor on the basis of its 10 percent allocation of resources and its list of previously submitted background jobs can make a prediction of the latest time the job will be terminated. The user, when notified of this termination time, can decide whether or not he wishes to avail himself this service, or perhaps ask for a quote on premium service. Clearly this technique can be extended to many price levels of premium service. In any case, the system then has only the problem of allocating 10 percent its resources to background jobs as time goes on. It should be obvious that resources and time are not equivalent but to a high degree of approximation the system can make such an equivalence when it has the ability of averaging over many users. Certainly the maximum amount of resource the system has available per unit time is fixed and there will be an average utilization factor. A mechanism for giving the background process 10 percent of the resources, is to place it in the scheduling algorithm with all other processes. Since the background process will quickly cascade to the bottom of the queue structure it will normally cease to get service. However the supervisor knowing that it must devote 10 percent of its time to this process should periodically extrapolate to the termination time and whenever it finds it is within a safety factor of missing the deadline, arbitrarily jump the priority level of the console-less process by 1. This will assure increased attention for the process and it should keep up. Note that

the background job does not require pre-emption, since it is not sensitive to minor fluctuations.

The third class of user is the guaranteed access user. An example of this type of use might be a radar tracking antenna which periodically needs a very short computation of its next azimuth. In this type of service the computer is valueless unless it can guarantee to provide a certain maximum response time to predictable size requests for computation.

The supervisor, when operating with one or more users of this type is careful at all times to insure that it has enough resources available to meet simultaneous demands by all users of guaranteed service. A user wishing to use this class of service must sign up in advance for it, since he may discover another user using a guaranteed service whose specifications conflict with his own.

The user negotiates with the system at sign up time, juggling his cost, the time at which the system is available, and his performance specifications. In general, he must tell the supervisor four things:

1. The time period he wishes to use the service.
2. The maximum amount of each system resource needed for a response.
3. The earliest time that (2) will be needed after the system is notified.
4. The minimum time between requests.

With this information, the system can examine its other commitments for service and quote availability and price to the user. Part of the policy involved here might be to commit not more than, say, 10% of the system resources to this class of service without special authorization.

At the time of actual usage of the guaranteed service, the user notifies the system that he wishes to use his sign-up privileges, and the system from then on assures without fail that resources are available to meet this commitment. A guaranteed access process, appears at the high end of the highest priority queue, and does not normally cascade down in priority unless, for example, it uses more resource (2) than was predicted and a shut off quota was not set. (Perhaps for simplicity, the shutoff quota is rigidly set to the amount of guaranteed system resource (2) above.)

Two special qualifications for this type of service must be noted:

1. It may be very expensive.
2. The user may discover that his computation is actually done earlier than the deadline he has set. He should be prepared to accept results early if necessary.

System Partitioning

The above survey is meant to give an overall philosophy for the specific implementation of resource allocation within the time-sharing system. There is in addition one larger issue which needs to be included in such a mechanism. This is the issue of partitioning both on a physical and a logical basis. (We have already discussed one simple partitioning problem connected with page sizes.) Clearly, it is relatively straight-forward to develop a physical partitioning mechanism wherein some processors, memory modules, etc., are isolated from the remainder and utilized separately. However, it must be noted that because of the high dependence on secondary storage this may not be a trivial problem. For example, one can remove a processor from the system very quickly, but removing a disk storage unit from the secondary storage hierarchy may require 15

minutes of transferring of the files stored there. The logical partitioning, which may or may not be done on a physical basis too, is quite important in the continuous operation of the system. This is especially so because it is becoming essential that all system programming be done on-line while most of the system continues to operate. Thus to check out sensitive areas of the supervisor such as scheduling, etc., it will be necessary to operate a second time-sharing system which is highly independent of the normally used one.